A Software Architecture for Knowledge-Based Systems

Fensel, D.A.; Groenboom, R.

published in
Knowledge Engineering Review
1999

DOI (link to publisher)
10.1017/S0269888999142097

document version
Publisher's PDF, also known as Version of record

The Knowledge Engineering Review, Vol. 14:2, 1999, 153–173
Printed in the United Kingdom

A software architecture for knowledge-based systems

DIETER FENSEL[1] and RIX GROENBOOM[2]

[1]*University of Karlsruhe, Institut AIFB, D-76128 Karlsruhe, Germany. Email: fensel@aifb.uni-karlsruhe.de*
[2]*University of Groningen, Department of Computing Science, P.O. Box 800, 9700 AV Groningen, The Netherlands. Email: rix@cs.rug.nl*

Abstract

The paper introduces a software architecture for the specification and verification of knowledge-based systems combining conceptual and formal techniques. Our focus is component-based specification enabling their reuse. We identify four elements of the specification of a knowledge-based system: a task definition, a problem-solving method, a domain model, and an adaptor. We present algebraic specifications and a variant of dynamic logic as formal means to specify and verify these different elements. As a consequence of our architecture we can decompose the overall specification and verification task of the knowledge-based systems into subtasks. We identify different subcomponents for specification and different proof obligations for verification. The use of the architecture in specification and verification improves understandability and reduces the effort for both activities. In addition, its decomposition and modularisation enables *reuse* of components and proofs. Therefore, a knowledge-based system can be built by combining and adapting different reusable components.

1 Introduction

Knowledge is *situated* and its usefulness for different situations is limited. The *brittleness* of Knowledge-Based Systems (KBSs) creates serious problems when trying to reuse it outside the context in which it evolved. When developing a knowledge model for a single application, the developer may have a good intuition about which assumptions can be made so as to deal with his problem adequately. In this case, hidden assumptions become apparent in cases where the system fails. The KBS may not be able to process a given input, or it returns a result that is not the solution as it is required. Using error situations and system breakdowns is the most common (implicit) search method for assumptions. However, this "method" may also cause significant damage. Reliable systems require the explication and formalization of their assumptions as an explicit part of their development process. In addition, contexts have the problematic feature that they change over time. As a consequence, the knowledge-based system must be maintained. A specification of the functionality of the systems and the assumptions it makes are essential for answering the questions as to whether it is necessary to change the system and how this can be done without losing other necessary properties. Finally, the problem of context-dependency is immediately present for knowledge models which are intended to be sharable and reusable. In this case they cannot be designed to intuitively fit to a given context because they must be applicable to a broad range of problems not known beforehand.[1]

During the last few years, several conceptual and formal specification techniques for KBSs have

[1]For a discussion of the general pros and cons of using formal methods, see van Harmelen and Fensel (1995).

been developed (see Studer et al. (1998), Fensel and van Harmelen (1994) and Fensel (1995c) for surveys) to deal with these problems. The main advantage of these modelling or specification techniques is that they enable the description of a KBS independent of its implementation. This has two main implications. First, such a specification can be used as a gold standard for the validation and verification of the implementation of the KBS. It defines the requirements the implementation must fulfil. Secondly, validation and verification of the functionality, the reasoning behaviour, and the domain knowledge of a KBS is already possible during the early phases of the development process of the KBS. A model of the KBS can be investigated independently of aspects that are only related to its implementation. Especially if a KBS is built up from reusable components, it becomes an essential task to verify whether the assumptions of such a reusable building block fit to each other, and to the specific circumstances of the actual problem and knowledge.

In the paper, we discuss a conceptual and formal framework for the specification of KBSs *composed of reusable building blocks*. Our conceptual framework is derived from the CommonKADS model of expertise (see Schreiber et al. (1994)) because this model has become widely used by the knowledge engineering community. We refined this model to describe KBSs that are built up by combining and adapting different components. The formal techniques applied are based on combining variants of algebraic specification techniques (see Bidoit et al. (1991) and Wirsing (1990)) and dynamic logic (see Harel, 1984). As a consequence of our modularised specification, we identify several proof obligations that arise to guarantee a consistent specification. The overall verification of a KBS is broken down into different types of proof obligations reducing the effort of the overall proof process, and enabling reuse of proofs as consequence of reusing components.

Our conceptual and formal model can be viewed as a software architecture for a specific class of systems, i.e. KBSs. A software architecture decomposes a system into components and defines their relationships (cf. Garlan and Perry (1995) and Shaw and Garlan (1996)). This recent trend in software engineering works on establishing a more abstract level in describing software artefacts than was common before. The main concern of this new area is the description of generic architectures that describe the essence of large and complex software systems. Such architectures specify classes of application problems instead of focusing on the small and generic components from which a system is built up. In the comparison section of this paper, we will take a closer look on analogies and differences between our work and this recent line of research in software engineering.

The paper develops ideas introduced in Fensel et al. (1996b). This is achieved by adding details and formal strength and by focusing on one of its various aspects. Basically, Fensel et al. (1996b) introduced the ideas of the four component architecture as presented here, the use of KIV (cf. Reif (1995)) for verifying such architectural specification, the use of assumptions for characterising and developing knowledge-based systems, and a method, eventually called *inverse* verification, for finding and constructing such assumptions. These various lines of research have been further worked out in a number of related papers covering the different aspects in more detail and precision. Most of them were triggered by a case study in parametric design problem-solving (Poeck et al., 1996) and its analysis in Fensel (1995a).

Here we focus on the different specification elements and their corresponding proof obligations. That is, we focus on the overall system architecture, but not on the process that establishes or uses such an architecture. The work we present here introduces the backbone that is used by other publications that focus on its specific aspects. The relationships of this paper to other publications of the authors is as follows:

- Fensel and van Harmelen (1994), Fensel (1995b, 1995c), van Harmelen and Fensel (1995), Angele et al. (1998) and Fensel et al. (1998a) describe pioneering work on *how to combine conceptual models of knowledge-based systems with formal specifications*. In this paper, we build on this work, however, refine some of the formal means and introduce a component-based framework (i.e. conceptual model) enabling knowledge and software reuse.
- Our architecture makes use of *some refinements of existing formal methods* that were introduced by the languages MLPM (Fensel & Groenboom, 1996) and MCL (Fensel et al., 1998b) that

provide a more suitable means for specifying the dynamic aspects of the reasoning process of knowledge-based systems (as worked out in Groenboom (1997)). An analysis of existing approaches for specifying these dynamics can be found in van Eck et al. (1998).

- The use of our framework for structuring the *verification process* of knowledge-based systems is described in Fensel and Schönegge (1997) and Fensel et al. (1998c). We follow and refine a proposal made by van Harmelen and Aben (1996).
- Describing reusable components requires the explicit formalization of their underlying *assumptions*. Otherwise, reusing a component in a new context may cause serious damage. Therefore, assumptions and explicit adaptation play a prominent role in our architecture. More detailed material on assumptions of problem-solving methods and their role and purpose in systems development and characterisation can be found in Fensel (1995a), Benjamins et al. (1996), Fensel and Straatman (1996, 1998) and Fensel and Benjamins (1998b). Fensel and Schönegge (1998) introduce the *inverse verification* method as an approach for finding such assumption in a semi-automatic manner.
- The *development process* of knowledge-based systems that makes use of our architecture (i.e., the architectural guidelines of our methodology) is described in Fensel et al. (1996a, 1997a, 1997), Fensel and Motta (1998) and Fensel and Schönegge (1998). Still, there is further work required to use our approach in practice.
- The architecture described in this paper currently forms the basis for the Unified Problem-solving Method development Language (UPML) (cf. Fensel et al. (1999a, 1999b)). UPML refines the generic adapter concept by distinguishing *refiners* (that take one specification element as input and provide an adapted version as output) and *bridges* (that connect two different component types). In addition, UPML introduces ontologies as a separate specification element that supports the reusable specification of data structures. UPML specifications are used by a internet-based brokering service (Fensel, 1997b; Benjamins et al., 1998; Fensel & Benjamins, 1998a) to support the combination, adaptation and distributed execution of knowledge components from different libraries.

In this paper, we focus on the different *products* of the specification and verification process. The discussion of the *process* of establishing such specifications, the different *roles* of the system developer (i.e., domain expert, knowledge engineer, programmer, etc.) as well as the available *tool* support in verification is beyond the scope of the paper. That is, this paper provides the backbone for a large number of other papers that extend it in different directions. In general, a software architecture covers the following aspects:

- a component model for decomposing the system description;
- architectural constraints that constrain meaningful combinations of components;
- and a process model (called design guidelines) that guide the process of developing systems following a specific component model.

Our paper focuses on the first aspect and sketches out the second one. The third aspect (i.e. the process model) is still under development, and we refer the reader to Fensel et al. (1996a, 1997a, 1997), Fensel and Motta (1998) and Fensel and Schönegge (1998).

The paper is organised as follows. In section 2, we discuss the different conceptual elements of a specification of a KBS, and in section 3 we sketch the different kinds of proof obligations that arise in our context. Section 4 introduces our formal means to specify the different elements using a simple example for illustrating the formalisations. Section 5 compares our architecture with related work, and Section 6 summarises the paper and defines objectives for future research.

2 A formal framework for the specification of knowledge-based systems

Our framework for describing a KBS consists of four elements (see Figure 1): a *task* that defines the problem that should be solved by the KBS; a *Problem-Solving Method* (PSM) that defines the

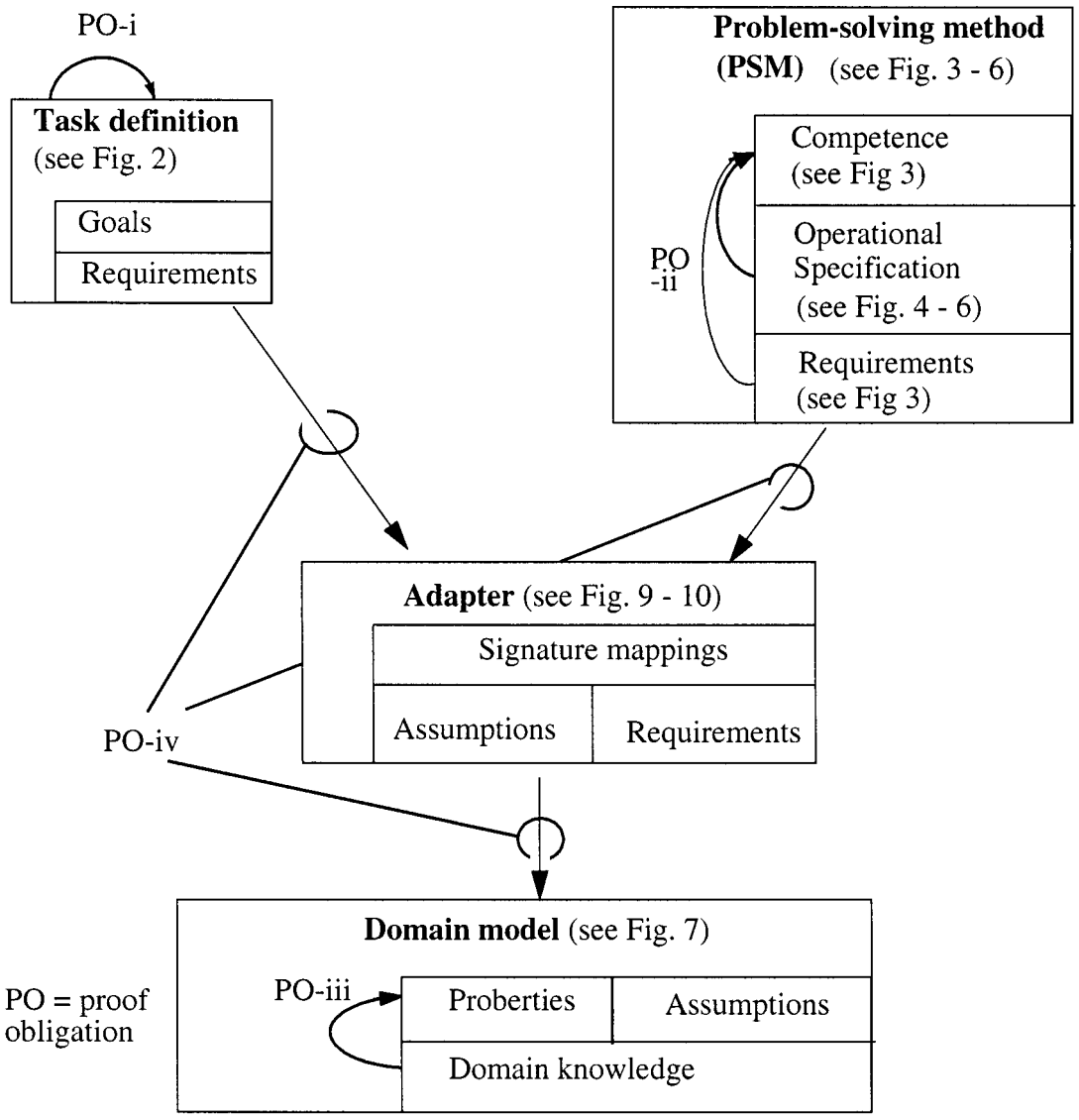

Figure 1 The four elements of a specification of a KBS.

reasoning process of a KBS; and a *domain model* that describes the domain knowledge of the KBS. Each of these three elements are described independently to enable the reuse of task descriptions in different domains (see Breuker & Van de Velde, 1994), the reuse of PSMs for different tasks and domains (Puppe, 1993; Breuker & Van de Velde, 1994; Benjamins, 1995), and the reuse of domain knowledge for different tasks and PSMs (cf. Gruber (1993), Top and Akkermans (1994), van Heijst et al. (1997)). Therefore, a fourth element of a specification of a KBS is an *adapter* that is necessary to adjust the three other parts to each other and to the specific application problem. This new element is used to introduce assumptions and to map the different terminologies.

2.1 The task

The description of a *task* specifies goals that should be achieved to solve a given problem. A second part of a task specification is the definition of requirements on domain knowledge. For example, a task that defines the derivation of a diagnosis requires causal knowledge explaining observables as domain knowledge. Axioms are used to define the requirements on such knowledge. A natural candidate for the formal task definition are *algebraic specifications*. They have been developed in

software systems (cf. Bidoit et al. (1991), Wirsing (1990)) and have already been applied by Spee and in't Veld (1994) and Pierret-Golbreich and Talon (1996) for KBS. In a nutshell, algebraic specifications provide a signature (consisting of types, constants, functions and predicates) and a set of axioms that define properties of these syntactical elements.

2.2 The problem-solving method

The concept of a *problem-solving method* (PSM) is present in many current knowledge-engineering frameworks (e.g., GENERIC TASKS (Chandrasekaran, 1986); ROLE-LIMITING METHODS (Marcus, 1988; Puppe, 1993); KADS (Breuker & Van de Velde, 1994) and CommonKADS (Schreiber et al., 1994); the METHOD-TO-TASK approach (Eriksson et al., 1995); COMPONENTS OF EXPERTISE (Steels, 1990); GDM (Terpstra et al., 1993); MIKE (Angele et al., 1998)). Libraries of PSMs are described in Benjamins (1995), Breuker and Van de Velde (1994), Chandrasekaran et al. (1992), Motta and Zdrahal (1996) and Puppe (1993). In general, a PSM describes which reasoning steps and which types of knowledge are needed to perform a task. Besides some difference between the approaches, there is strong consensus that a PSM:

- decomposes the entire reasoning task into more elementary inferences;
- defines the types of knowledge that are needed by the inference steps to be done; and
- defines control and knowledge flow between the inferences.

In addition, Van de Velde (1988) and Akkermans et al. (1993) define the *competence* of a PSM independently of the specification of its operational reasoning behaviour. A competence of a PSM is a logical theory that characterises the solutions provided by it. It corresponds to what is called a specification of the functionality in software engineering. Proving that a PSM has some competence has the clear advantage that the selection of a method for a given problem and the verification of whether a PSM fulfils its task can be done independently from details of the internal reasoning behaviour of the method.

The description of a PSM consists of three elements in our framework: a competence description, an operational specification, and requirements on domain knowledge.

The definition of the functionality of the PSM introduces the *competence* of a PSM independently from its dynamic realisation. As for task definitions, algebraic specifications can be used for this purpose.

An *operational description* defines the dynamic reasoning of a PSM. Such an operational description explains how the desired competence can be achieved. It defines the main reasoning steps (called *inference actions*) and their dynamic interaction (i.e., the knowledge and control flow) to achieve the functionality of the PSM. We use a variant of dynamic logic (cf. Fensel et al., 1998b) to express procedural control over the execution of inferences. The definition of an inference step could recursively introduce a new (sub-)task definition. This process of stepwise refinement stops when the realisation of such an inference is regarded as an implementation issue that is neglected during the specification process of the KBS.

The third element of a PSM introduces *requirements* on domain knowledge. Each inference step, and therefore the competence description of a PSM, requires specific types of domain knowledge. These complex requirements on domain knowledge distinguish a PSM from usual software products. Pre-conditions on valid inputs are expanded to complex requirements on available domain knowledge. Again, we will apply abstract data types for the specification of the requirements of a PSM.

The competence description of the PSM as well as the task definition are declarative specifications. The former specifies the actual functionality of the KBS (given that the domain knowledge fulfils the requirements of the PSM), and the latter specifies the problem that should be solved by applying the KBS. We make a distinction between both for two reasons:

- First, a PSM introduces requirements on domain knowledge in addition to the task definition.

This knowledge is not necessary to define the problem, but is required to describe the solution process of the problem.

- Secondly, we cannot always assume that the functionality of the KBS is strong enough to completely solve the problem. Most problems tackled with KBSs are inherently complex and intractable (cf. Bylander et al. (1991) and Motta and Zdrahal (1996)). PSMs needs to introduce assumptions that reduce the problem to a size they can deal with (see Fensel and Straatman (1998)). The later discussed adapters are the specification elements that contain the assumptions that have to be made to bridge the gap between both specifications.

A simple example may clarify these two points. The task of finding a global optimum is defined in terms of a preference relation. First, a PSM based on a local search technique requires in addition a local neighbourhood relation to guide the search process. This knowledge is not necessary to define the task but to define the problem-solving process and its competence. Depending on the properties of this neighbourhood relation, different competences of a method are possible (cf. Gamma et al. (1995) and Fensel and Schönegge (1998)). Secondly, the task of finding an optimal solution could easily define an NP-hard problem. The PSM based on a local search technique may provide solutions in polynomial time. However, it derives only a local optimum. Therefore, one must either put strong requirements on domain knowledge (each local optima must also be a global one) or one must weaken the task to local instead of global optima (cf. Fensel and Benjamins (1998b)) to establish the correspondence of PSM and task.

For the same reasons, we include a specification of the reasoning *process* of the PSM (called operational specification). In software engineering, the distinction between a functional specification and the design/implementation of a system is often discussed as a separation of *what* and *how*. During the specification phase, *what* the system should do is establish in interaction with the users. *How* the system functionality is realised is defined during design and implementation (e.g., which algorithmic solution can be applied). This separation—which even in the domain of software engineering is often not practicable in the strict sense—does *not* work in the same way for KBSs: a large amount of the problem-solving knowledge, i.e., knowledge about *how* to meet the requirements, is not a question of efficient algorithms and data structures, but exists as heuristics as a result of the experience of an expert. For many problems which are completely specifiable, it is not possible to find an efficient algorithmic solution. Often they are easy to specify, but it is not necessarily possible to derive an efficient algorithm from these specifications; heuristics and domain-specific inference knowledge are needed for the efficient derivation of a solution. One must not only acquire knowledge about what a solution for a given problem is, but also knowledge about how to derive such a solution in an efficient manner. Already at the knowledge level there must be a description of the domain knowledge and the problem-solving method which is required by an agent to solve the problem effectively and efficiently. In addition, the symbol level has to provide a description of efficient algorithm solutions and data structures for implementing an efficient computer program. As in software engineering, this type of knowledge can be added during the design and implementation of the system. Therefore, a specification language for KBSs must *combine non-functional and functional specification techniques*: on the one hand, it must be possible to express algorithmic control over the execution of substeps. On the other hand, it must be possible to characterise the overall functionality and the functionality of the substeps (i.e., the inference actions) without making commitments to their algorithmic realisations.

2.3 The domain model

The description of the *domain model* introduces the domain knowledge as it is required by the PSM and the task definition. Ontologies are proposed in knowledge engineering as a means to represent domain knowledge in a reusable manner (cf. Gruber (1993), Top and Akkermans (1994) and van Heijst et al. (1997)). Our framework provides three elements for defining a *domain model*: a meta-

level characterisation of *properties* of the domain model, *assumptions* of the domain model, and the *domain knowledge* itself.

The *domain knowledge* is necessary to define the task in the given application domain and necessary to carry out the inference steps of the chosen problem-solving method. Properties and assumptions differ in their state of truth. Properties can be derived from the domain knowledge, whereas assumptions are properties that have to be assumed to be true, i.e., they have not been proven or they cannot be proven. Assumptions capture the implicit and explicit assumptions made while building a domain model of the real world. *Properties* and *assumptions* are both used to characterise the domain knowledge. They are the counterpart of the requirements on domain knowledge introduced by the other parts of a specification. Some of these requirements may be directly inferred from the domain knowledge (and are therefore properties of it), whereas others can only be derived by introducing assumptions about the environment of the system and the actually provided input. For example, typical external assumptions in model-based diagnosis are: the fault model is complete (no fault appears that is not captured by the model); the behavioural description of faults is complete (all fault behaviours of the components are modelled), the behavioural discrepancy that is provided as input is not the result of a measurement fault, etc. (cf. Fensel & Benjamins, 1998b).

2.4 The adapter

Adapters are of general importance for component-based software development. Gamma et al. (1995) introduce an adapter pattern in their textbook on design patterns for object-oriented system development. Such adapters enable reusable descriptions of objects, and make it possible to combine objects that differ in their syntactical input and output descriptions. In our architecture, adapters are used to introduce further requirements and assumptions that are needed to relate the competence of a PSM to the functionality given by the task definition (cf. Fensel (1995a) and Fensel and Benjamins (1998b)). We have already mentioned the fact that an adapter usually introduces new requirements or assumptions because in general, most problems tackled with KBSs are inherently complex and intractable. A PSM can only solve such tasks with reasonable computational effort by introducing assumptions that restrict the complexity of the problem or by strengthening the requirements on domain knowledge. Task, PSM and domain model can be described independently and selected from libraries because adapters relate the three other parts of a specification together and establish their relationship in a way that meets the specific application problem. Their consistent combination and their adaptation to the specific aspects of the given application—since they should be reusable they need to abstract from specific aspects of application problems—must be provided by the adapter.

3 The main proof obligations

Following the conceptual model of the specification of a KBS, the overall verification of a KBS is broken down into four kinds of proof obligations (see Figure 1):

(PO-i) **Task consistency**. The consistency of the task definition ensures that a model exists. Otherwise, we would define an unsolvable problem. The requirements on domain knowledge are necessary to prove that the goal of the task can be achieved. Such a proof is usually achieved by constructing a model via an (inefficient) generate and test like implementation.

(PO-ii) **PSM consistency**. We have to show that the operational specification of the PSM describes a PSM for which termination can be guaranteed,[2] and that the PSM has the

[2]In fact, that excludes non-terminating methods like those used for monitoring tasks from our framework. For such tasks, a richer formal framework based on temporal logic would be required. In our case, we assume terminating programs that can be represented as input-output pairs.

competence as specified. This proof obligation recursively returns for each non-elementary inference action of a PSM. In addition to termination, one may also want to include some thresholds for the efficiency of the method by including it as part of the competence description (cf. Shaw (1989), Fensel and Schönegge (1998) and van Harmelen and ten Teije (1998)).

(PO-iii) **Domain consistency**. We have to ensure internal consistency of the domain model. The overall domain knowledge does not need to be consistent, but it must be dividable into consistent parts. In addition, we have to prove that given its assumptions the domain knowledge actually implies its meta-level characterisation.

(PO-iv) **Overall system consistency**. We have to establish the relationships between the different elements of the specification:

(a) **APT-consistency**. We have to prove that the requirements of the adapter imply the knowledge requirements of the PSM and the task.

(b) **PT-consistency**. In addition to the already existing requirements, an adapter may need to introduce new requirements on domain knowledge and assumptions (properties that do not follow from the domain model) to guarantee that the competence of the PSM is strong enough to process the task.

(c) **AD-consistency**. We have to prove that the requirements of the adapter are implied by the properties and assumptions of the domain model.

Notice that PO-i deals with task definition internally, PO-ii deals with the PSM internally, and PO-iii deals with the domain model internally, whereas PO-iv deals with the external relationships between task, PSM, domain knowledge and adapter. Thus, a separation of concerns is achieved that contributes to the feasibility of the verification (cf. van Harmelen and Aben (1996)). The conceptual model applied to describe KBSs is used to break the general proof obligations into smaller pieces, and makes parts of them reusable. As PSMs can be reused, the proofs of PO-ii do not have to be repeated for every application. These proofs have to be done only when a new PSM is introduced to the library. Similar proof economy can be achieved for PO-i and PO-iii by reusable task definitions and domain models. The application specific proof obligation is PO-iv.

Assumptions concerning the input cannot be verified during the development process of a KBS. However, their derivation is very important because they define preconditions for the validity of inputs that must be checked for actual inputs to guarantee the correctness of the system.

More details on the formalisation of these architectural constraints can be found in Fensel et al. (1999b).

4 The formal specification of the different components

In this section we characterise the different elements of a system architecture in more detail.

4.1 Formalising tasks

We use a simple task to illustrate our approach. The task *abductive diagnosis* receives a set of observations as input and delivers a complete and parsimonous explanation (e.g., see Bylander et al. (1991)). An explanation is a set of hypotheses. A *complete explanation* must explain all input data (i.e., *observations*) and a *parsimonious* explanation must be minimal (that is, no subset of hypotheses explains all *observation*). Figure 2 provides the task definition for our example. Any explanation that fulfils the *goal* must be *complete* and *parsimonious*. The input requirement ensures that there are *observations*.

The task does not introduce any requirements on domain knowledge by axioms but the domain model must provide sets to interpret the sorts *datum* and *hypothesis* and an explanation function *expl*. We will see how the signature mapping is achieved by the adapter.

task *complete and parsimonious explanation*
sorts
datum, data : **set of** *datum,*
hypothesis, hypotheses : **set of** *hypothesis*
functions
expl: *hypotheses* → *data*
observables: *data*
predicates
goal : *hypotheses*
complete: *hypotheses*
parsimonious: *hypotheses*
variables
x : *datum*
H,H' : *hypotheses*
axioms
goal
$\forall H\ (goal(H) \rightarrow complete(H) \wedge parsimonious(H))$
$\forall H\ (complete(H) \leftrightarrow expl(H) = observables)$
$\forall H\ (parsimonious(H) \leftrightarrow \neg\exists H'\ (H' \subset H \wedge expl(H) \subseteq expl(H')))$
input requirement
$\exists x\ (x \in observables)$
endtask

Figure 2 The task definition for *abduction*.

4.2 The problem-solving method

Finding a complete and parsimonious explanation is NP-hard in the number of hypotheses (Bylander et al., 1991). Therefore, we have to apply heuristic search strategies. In the following, we characterise a local search method which we call *set-minimiser*. The discussion of whether other methods would be more suitable or how we have selected this method is beyond the scope of this paper (see Fensel (1997a) and Fensel and Motta (1998) for more details). In the following, we first provide the black box specification of the method. That is, we specify the competence provided by the method and the knowledge required by the method. Then, we provide a white box specification of the operational reasoning strategy, which explains how the competence can be achieved. The former is of interest during reuse of PSMs whereas the latter is required for developing PSMs.

4.2.1 The black box description: competence and requirements

The competence description of the PSM as well as the task definition are declarative specifications. In consequence, we apply the same formal means for their specifications: the task specifies the problem that should be solved by applying the KBS and the PSM specifies the actual functionality of the KBS (given that the domain knowledge fulfils the requirements of the PSM). A PSM introduces additional requirements on domain knowledge and may weaken the task definition. However, both aspects can directly be covered by algebraic data types.

The competence theory in Figure 3 defines the competence that we call *set-minimiser*. *Set-minimiser* is able to find a correct and locally minimal set. Local minimality means, that there is no correct subset of the output that has only one less element. The method has one requirement: it must receive a correct initial set (cf. Figure 3). The competence as well as the requirements illustrate the additional aspects that are introduced by the PSM:

- The task of finding a parsimonious set is reduced to local parsimonious sets.
- Constructing an initial correct set is beyond the scope of the method. It is assumed as being

competence *set-minimiser*
sorts *object, objects* **set of** *object*
predicates *correct* : *objects*
variables *x* : *object*
constants *Input, Output* : *objects*
axioms
$correct(Output)$
$\neg \exists x\ (x \in Output \wedge correct(Output \setminus \{x\}))$
endcompetence

requirements *set-minimiser*
sorts *object, objects* **set of** *object*
predicates *correct* : *objects*
constants *Init* : *objects*
axioms
$correct(Init)$
endrequirements

Figure 3 The competence and requirements of the PSM.

provided by the domain knowledge, by a human expert or by another PSM. The method only minimises this correct set.

The PSM does not make any assumptions what a *correct* set is. This has to be determined by the domain a PSM is instantiated for. In general, a PSM could add additional axioms that restrict the valid interpretations of a domain theory. The predicate *correct* can be understood as a generic parameter that will be replaced by an actual parameter when connecting the PSM to a domain.

4.2.2 The operational specification

Our method *set-minimiser* uses depth-first search through a search space that it structured by set inclusion. The entire method is decomposed into the following two steps: the inference action *generate* generates all successor sets that contain one less element. The inference action *select* selects one correct set from the successors and the predecessor. The inference structure of this method is given in Figure 4 roughly following the conventions of CommonKADS (Schreiber et al., 1994). It specifies the main inferences of a method (i.e., its substeps), the dataflow between the inferences (i.e., the knowledge flow and the dynamic knowledge roles) and the knowledge types (i.e., the static knowledge roles) that are required by them. In the following, we will define each of these elements in more detail. In addition, we will have to define the control flow between the inference steps. The latter introduces a strong new requirement on our means for formalisation requiring a logic of changes.

Inference actions and knowledge Roles

Again we use algebraic specifications to specify the functionality of inference actions and knowledge roles. Dynamic knowledge roles (dkr) are means to represent the state of the reasoning process and axioms can be used to represent state invariants. They correspond roughly to state schemas in Z (Spivey, 1992). The interpretation of constants, functions and predicates may change during the problem-solving process. Static knowledge roles (skr) are means to include domain knowledge into the reasoning process of a problem-solving method. Our method *set-minimiser* requires knowledge about correct sets and an initial set. This is modelled by the static knowledge roles. Figure 5 provides the definitions of the two inference actions and the knowledge roles. Basically, *generate* derives all subsets that have one element less and *select* selects a successor (one of these reduced sets) if a correct successor exists. Otherwise its selects the original node.

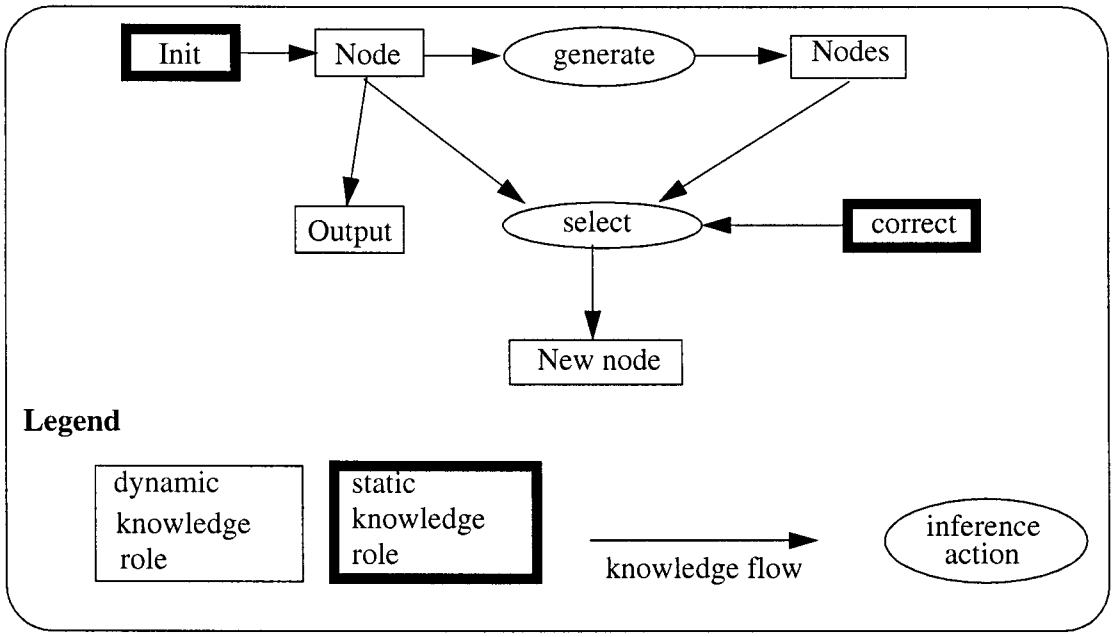

Figure 4 Knowledge flow diagram of *set-minimiser*.

inf *generate*
sorts *object, objects* **set of** *object*
variables z : *object*, x,y : *objects*
axioms
$generate\ (x,y) \leftrightarrow \exists z\ (z \in y \wedge x = y \setminus \{z\})$
endinf

inf *select*
sorts *object, objects* **set of** *object, objects set* **set of** *objects*
variables x,y,z' : *objects*, z : *objects set*
axioms
A correct successor is selected
$\forall x,y,z\ (\exists z'\ (z' \in z \wedge correct(z') \rightarrow select(x,y,z) \wedge correct(x) \wedge x \in z)$
No correct successor exists and the old node is selected
$\forall x,y,z\ (\neg\exists z'\ (z' \in z \wedge correct(z')) \rightarrow select(x,y,z) \wedge correct(x) \wedge x = y)$
endinf

---

dkr *New node*
sorts *object, objects* **set of** *object*
constant *New node* : *objects*
enddkr

dkr *Node*
sorts *object, objects* **set of** *object*
constant *Node* : *objects*
enddkr

dkr *Nodes*
sorts *object, objects* **set of** *object*
predicate *Nodes* : *objects*
enddkr

skr *Correct*
sorts *object, objects* **set of** *object*
predicates *correct* : *objects*
endskr[1]

skr *Init*
sorts *object, objects* **set of** *object*
constant *Init* : *objects*
endskr

1. Sets can be either represented by constants or predicates.

Figure 5 The specification of the inference actions and the knowledge roles.

```
Node := Init;
if Node = ∅
  then Output := ∅
  else
    repeat
      Nodes := λx.generate(x,Node);
      ∪x.((select(x,Node,z) ∧ Nodes(z))?; New node := λy.(x=y)
    until New node = Node
    Output := Node
endif
```

Figure 6 The specification of dynamics.

Control flow

The operational description of a PSM is completed by defining the control flow (see Figure 6) that defines the execution order of the inference actions. The specification in Figure 6 uses the *Modal Change Logic (MCL)* (Fensel et al., 1998b), which was developed in to combine functional specification of substeps with procedural control over them. MCL is a generalised version of the Modal logic of Creation and Modification (MLCM, see Groenboom and Renardel de Lavalette (1994) and Groenboom (1997)) and the Modal Logic for Predicate Modification (MLPM) (Fensel & Groenboom, 1996). Each of these languages are variants of dynamic logic. Dynamic logic and (Harel, 1984) was developed to express states, state transitions, and procedural control of these transitions in a logical framework. Dynamic logic uses the *possible-worlds* semantics of Kripke (1959) for this purpose. A state is represented by a possible world through the value assignments of the program variables. MCL extends the representation of a state. A state is represented by an *algebra* following the states-as-algebras paradigm of abstract state machines (Gurevich, 1994). A state transition is achieved by changing the truth values of a predicate or the values of a term. MCL provides the usual procedural constructs such as sequence, if-then-else, choice and while-loop to define complex transition.

A complete introduction to MCL is beyond the scope of this paper (see Fensel et al. (1998b) and Fensel and Groenboom (1996) for more details). We will only mention some features required to understand our example. Dynamic logic is often presented with the variable assignments $x := t$ as its atomic programs and value assignments of a number of free variables as state representation. MCL provides a more structured state representation. The interpretation of constants, function and predicates model a state. For example, an array can be model by an unary function *array* where *array*(*i*) denotes the *i*-th element of the array. In different states, *array*(*i*) may refer to different values. MCL provides more powerful constructs to express an inference of a KBS as an elementary state transition. For example, in MCL we have

$$f := \lambda x.t \quad \text{and} \quad p := \lambda x.A$$

that changes the interpretation of a function or predicate such that $f(x) = t(x)$ and $p(x) = A(x)$, and

$$\cup x.\alpha$$

that performs α for a non-deterministically chosen x for a given program α. This is necessary to implement an inference and return non-deterministically one of its solutions. These atomic programs generalise the state transitions as introduced by the KADS specification languages KARL (Fensel et al., 1998a) and $(ML)^2$ (van Harmelen & Balder, 1992). Besides the atomic programs, MCL has the normal imperative statements for sequential composition, choice and repetition.

In Figure 6, we apply $p := \lambda x.A$ to express that *Nodes* is updated by all successor sets of the set contained by *Node* and $\cup x.\alpha$ to express that *New node* is updated (non-deterministically) by one correct successor set if it exists or the predecessor if not.

4.3 The domain model

A domain model consists of three main parts: the domain knowledge, its properties, and its assumptions. In addition, a signature definition is provided that defines the common vocabulary of the other three elements.

The medical domain model we have chosen for our example is a subset of a large case-study in the formalisation of domain knowledge for an anesthesiological support system. The support system should diagnose a (limited) number of hypotheses, based on real-time acquired data. This data is obtained from the medial database system Carola (de Geus & Rotterdam, 1992), which performs on-line logging of measurements. The formal model includes knowledge of how to interpret these raw measurements (quantitative data) and causal relations between qualitative data (see Renardel de Lavalette et al. (1997) for details).

Some of the simplifications we have made include:

- In this simplified domain we have a "one step" causal relation R. In practice we have the transitive and irreflexive closure $R+$. Note that we do not have the reflective transitive closure R*, since a symptom cannot be a hypothesis for itself. In a more complex version of this domain model, a symptom can be the hypothesis for another symptom. This kind of reasoning is left out for expository reasons (see Renardel de Lavalette et al. (1997) for details).
- To obtain a complete model, we restricted the number of possible hypotheses. Although this seems a major restriction, it is the same as we have to employ to the larger knowledge base. In consultation with the physician, we restrict the number of diagnoses (hypotheses) we want the system to detect. Then we design a system to detect these hypotheses, leaving the final diagnosis to the physician. This is also the main reason why the system is a *support* system; the goal of the system is not to replace a physician, only to support him.
- A last simplification is the abstraction from time. Although the quantitative data is measures a certain time-points, we model causal-knowledge as non-temporal. The notion of time is handled elsewhere in the domain (see Groenboom (1997) and Renardel de Lavalette et al. (1997) for details).

In this domain we deal with abstract notions, derived from interpretations of measurements. The exact meaning of *HighPartm* (which stands for a High mean arterial blood-pressure) is defined elsewhere in the formal domain model (see Groenboom, 1997). Another technical term is *ToolowCOP*, which refers to a too low Cellular Oxygen Pressure. Figure 7 sketches the causal knowledge and Figure 8 defines the signature and axioms of our domain model. It contains hypotheses and symptoms and a causal relationship between them. The properties make explicit

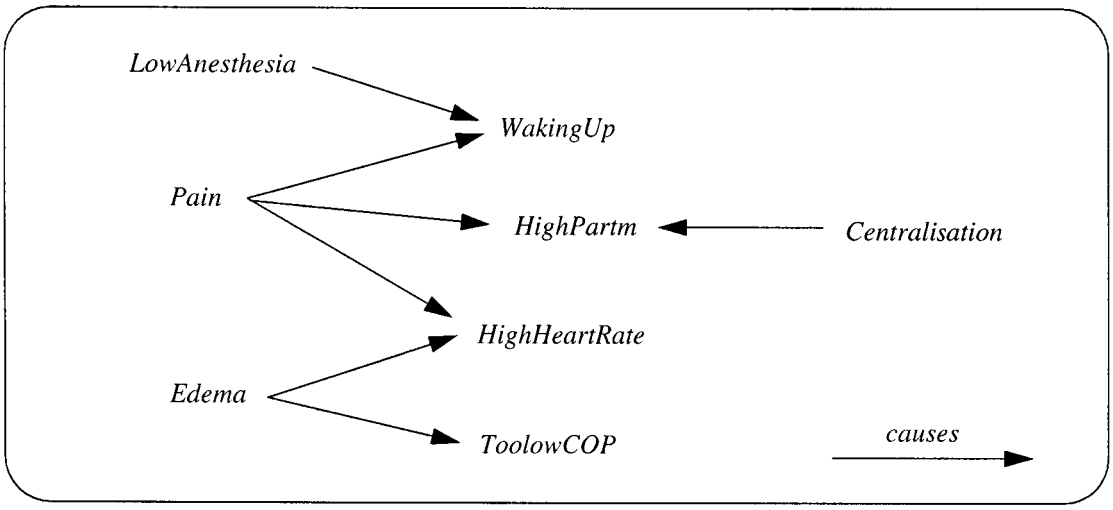

Figure 7 The domain knowledge.

domain model *anesthesiology*
signature
sorts *hypothesis, hypotheses* **set of** *hypothesis, symptom*
functions
HighHeartRate, HighPartm, ToolowCOP, WakingUp : *symptom*
Centralisation, Pain, Edema, LowAnesthesia : *hypothesis*
predicates
causes: *hypothesis* x *symptom*
variables
h : *hypothesis*
s : *symptom*
H, H': *hypotheses*
properties
there is a cause for each symptom
$\forall s \exists h\ causes(h, s)$
the fault knowledge is monotonic
$H \subseteq H' \rightarrow \{s \mid h \in H \wedge causes(h,s)\} \subseteq \{s \mid h \in H' \wedge causes(h,s)\}$
assumption
complete fault knowledge:
We know all possible causes of the provided manifestations.
$h = lowAnesthesia \wedge s = wakingUp \vee$
$h = centralisation \wedge s = highPartm \vee$
$h = pain \wedge s = highPartm \vee$
$h = pain \wedge s = wakingUp \vee$
$h = pain \wedge s = highHeartRate \vee$
$h = edema \wedge s = highHeartRate \vee$
$h = edema \wedge s = oolowCOP \vee$
$\vee \neg cause\ relation(h,s)$
domain knowledge
$causes(LowAnesthesia,WakingUp)$
$causes(Centralisation,HighPartm)$
$causes(Pain,HighPartm)$
$causes(Pain,WakingUp)$
$causes(Pain,HighHeartRate)$
$causes(Edema,HighHeartRate)$
$causes(Edema,ToolowCOP)$
enddm

Figure 8 The domain model.

that there is a cause for each symptom and hypotheses do not conflict. That is, different hypotheses do not lead to an inconsistent set of symptoms. In our domain this is guaranteed by the fact that we do not have knowledge about negative evidence (i.e., a symptom may rule out an explanation). Assuming more causes only leads to a larger set of symptoms that can be explained. The *complete-fault-knowledge assumption* guarantees that there are no other unknown faults like hidden diseases. Only under this assumption we can deductively infer causes from observed symptoms. However, it is a critical assumption when relating the output of our system to the actual problem and domain (cf. Fensel & Benjamins, 1998b).[3]

[3]Notice that we do not assume complete knowledge of symptoms.

adapter $TP_{Adapter}$
include *set-minimiser, complete and parsimonious explanation*
rename *set-minimiser* **by** *abduction*
object → *hypothesis, objects* →*hypotheses, correct* → *complete*
variables x : *datum*
proof obligation *goal*(*Output*)
requirements
$\exists x\ (x \in observables)$
complete(*Init*)
endadapter

Figure 9 The initial version of the $TP_{Adapter}$.

4.4 Two adapters

An adapter has to link the different signatures of task, PSM and domain, and has to add further axioms to guarantee their proper relationships. We use abstract data types for this purpose. First we demonstrate how to link task and PSM by the $TP_{Adapter}$. Then we discuss their relations with the domain model defined by the $D_{Adapter}$.

4.4.1 Connecting task and PSM

Combining task and PSM requires three activities: establishing of syntactical links between different terminologies by mapping (see Reif (1992) for more details), establishing of semantic links between different predicates, and the introduction of new assumptions and requirements to establish that the goals of the task are implied by the output of the method.

In our case study, we have to link the sort *object* and *objects* and the predicate symbol *correct* of the PSM by renaming. The appropriate interpretation of predicates has to be ensured by axioms if they cannot be linked directly. The necessity that the output of the method implies the goal of the task is stated as proof obligation (see Figure 9).

The $TP_{Adapter}$ contains the collection of the requirements introduced by task and PSM. This includes: any application problem provides at least one observation and the set of hypotheses delivered by *Init* must be a complete explanation of all observations (see Figure 9). These requirements must be fulfilled by the domain knowledge and the input to ensure that the task is well-defined and the inference steps of the PSM work properly.

Finally, we have to introduce new assumptions and requirements to ensure that the competence of the PSM implies the goal of the task (i.e., to fulfil the proof obligation of the adapter). We already know that *Output* contains a locally-minimal set. Each subset of it that contains one less element is not a complete explanation. Still this is not strong enough to guarantee parsimonity of the explanation in the general case. There may exist smaller subsets that are complete explanations. In Fensel and Schönegge (1998), we have proven that the global-minimality of the task definition is implied by the local-minimality if we introduce the *monotonic-problem assumption* (see Bylander et al. (1991)):

$$H \subseteq H' \rightarrow expl(H) \subseteq expl(H')$$

For details on how to find such assumptions with an interactive theorem prover see Fensel and Schönegge (1997, 1998).

Figure 10 provides the intermediate adapter that contains the fulfilled proof obligation and the new assumption. Whether the monotonicity property must be stated as an assumption or whether it can be formulated as a requirement on domain knowledge in the final version of the adapter can be decided when specifying the second aspect of the adapter, its connection with the domain knowledge.

adapter $TP_{Adapter}$
include *set-minimiser, complete and parsimonious explanation*
rename *set-minimiser* **by** *abduction*
object → *hypothesis, objects* →*hypotheses, correct* → *complete*
variables x : *datum*, H,H': *hypotheses*
axioms *goal*(*Output*)
requirements
$\exists x\ (x \in observables)$
complete(*Init*)
assumptions $\forall H, H'(H \subseteq H' \rightarrow expl(H) \subseteq expl(H'))$
endadapter

Figure 10 The intermediate version of the $TP_{Adapter}$.

adapter $D_{Adapter}$
include *anesthesiology*, $TP_{Adapter}$
rename *anesthesiology* **by** $TP_{Adapter}$
symptom → *datum*,
variables h : *hypothesis*, x : *datum*, H,H': *hypotheses*
axioms
$\forall x, H\ (x \in expl(H) \leftrightarrow \exists\ h\ (h \in H \wedge causes(h,x)))$
proof obligation
$\exists x\ (x \in observables)$
complete(*Init*)
$\forall H, H'(H \subseteq H' \rightarrow expl(H) \subseteq expl(H'))$
endadapter

Figure 11 The initial $D_{Adapter}$.

4.4.2 Connecting with the domain model

Finally, we have to link the domain model to the other components using the $D_{Adapter}$ (see Figure 11). We have to map the different terminologies, to define the logical relationships between domain knowledge and the other parts of the specification by axioms, and to prove the requirements on domain knowledge. For our example, most of these requirements follow straightforwardly from the meta-knowledge of the domain model. Therefore, the monotonicity of hypotheses can be stated as a requirement, because it follows from the specification of the domain knowledge. If a requirement cannot be derived from the domain knowledge it must be stated as an assumption. In our example, the requirement

$$\exists x\ (x \in observables)$$

cannot be derived from the domain knowledge because it is concerned with the input. However, assuming an input for deriving a diagnosis is not a critical assumption.[4] It remains to ensure that *Init*

[4]For example, a more serious assumption would be the single-fault assumption (cf. Davis (1984)). Formulating it as a requirement on domain knowledge enforces that each possible fault combination is represented as a single fault by the domain knowledge. Therefore, it is often used as an assumption that limits the scope of the problems that can be handled correctly by the system. Cases where a single fault is the actual cause can be solved correctly by the system. Situation with more complex error situations must be solved without support by the system. In general, formulating a property as a requirement increases the demand on domain knowledge and formulating a property as an assumption decreases the application scope of the system (cf. Fensel and Benjamins (1998b)).

delivers a correct set of hypotheses. An easy way to achieve this is to deliver the entire set of hypotheses (given the monotony of the problem), i.e., $\forall h \in Init$.

5 Related work

The CommonKADS model of expertise (Schreiber et al., 1994) provides a conceptual model for describing the different knowledge types of a KBS. It distinguishes three layers: domain layer, inference layer and task layer. The *domain layer* corresponds to the domain knowledge in our architecture. We add assumptions that describe the relation the domain knowledge has with the actual domain (i.e., which assumptions have been made during modelling a part of reality) and meta knowledge that provide properties of these knowledge. These properties are essential when connecting PSMs and tasks with a domain. The *inference layer* provides the inferences, knowledge roles and dataflow dependencies of a KBS. In our architecture they are part of the operational specification of a PSM. Again requirements (i.e., assumptions) and competence descriptions are added to these descriptions to support the reusability of such reasoning strategies. Finally, the task layer defines the goals a KBS should achieve, their decomposition into subgoals and the control that regulates the order of execution of the inferences. In our architecture the latter two aspects are part of the operational specification of a PSM, and for the task remains to define the goals that should be achieved by the KBS. Again, requirements are added to expect the sort of domains a task can be instantiated for.

Recently, the knowledge level (Newell, 1982) has been encountered in software engineering (cf. Garlan & Perry, 1994; Shaw & Garlan, 1996). Work on software architectures establishes a higher level to describe the functionality and the structure of software artefacts. The main concern of this new area is the description of generic architectures that describe the essence of large and complex software systems. Such architectures determine specific classes of application problems, instead of focusing on the small and generic components from which a system is built up. Our conceptual model fits into this recent trend. It describes an architecture for a specific class of systems: knowledge-based systems. Usually, architectures are described by their components and connectors that establish the proper relationships between the former. In our case, we have three types of components (tasks, problem-solving methods and domain models) and adapters that connect them. However, adapters do no deal with communication aspects as it is often the case with connectors (cf. Yellin & Strom, 1997).

Work on formalising software architectures characterises the functionality of architectures in terms of assumptions on the functionality of its components (Penix and Alexander, 1997; Penix et al., 1997). This shows strong similarities to our work where we define the competence of a problem-solving method in terms of assumptions on domain knowledge (which can be viewed as one or several components of a knowledge-based system) and the functionality of elementary inference steps. However, Penix and Alexander (1997) and Penix et al. (1997) abstract from the operational specification of the architecture and keep its specification and verification separate. In our framework, this is treated as an integrated piece of the specification of the entire architecture. This is the reason why we rely on a combination of algebraic specifications and dynamic logic for specification and verification, whereas Penix and Alexander (1997) and Penix et al. (1997) use only algebraic specifications.

An interesting architecture for the specification of problem-solving methods in the area of model-based diagnosis is presented by ten Teije (1997). The specification of the competence of a problem-solving method is parameterised by a fixed set of component types. Changing the functionality of a component by selecting a different instantiation for one of the component types modifies the competence of the entire method. While it is a very interesting approach, we still wonder whether it is really useful for specifying problem-solving methods. In our opinion, two key features of problem-solving methods are missing:

- Problem-solving methods describe how a pattern is solved, and the operational strategy is not covered by the declarative specification of its competence.
- A task introduces knowledge requirements to define a problem in domain-specific terms. A problem-solving method introduces additional knowledge requirements that are necessary to solve the problem. A local search method needs a local structure that is not necessary to define the problem but to define the problem-solving process. This type of knowledge is not present as a parameter in her framework.

In consequence, we think ten Teije (1997) presents an interesting framework for a parameterised specification of model-based diagnosis *tasks* rather than problem-solving methods.

6 Conclusions and future work

In the paper, we introduce a formal and conceptual framework for specifying and verifying knowledge-based systems. One can specify tasks, problem-solving methods, domain models and adapters and can verify whether the assumed relationships between them are guaranteed, i.e., which assumptions are necessary for establishing these relationships. Such an architecture improves the understandability of specification and verification. The modularisation reduces the effort of specification and verification by defining smaller contexts and enabling reuse of smaller parts in new contexts. The idea of an adapter allows to combine and adapt reusable elements without being forced to modify them. The specification of the problem-solving methods is decomposed into external and internal aspects. The specification of the competence of the problem-solving method provides all necessary aspects for relating it to the task that must be carried out. When specifying a reusable problem-solving method it must be proven once, whether the operational specification specifies a computational process that has the specified competence. When reusing the method, it is possible to abstract from all details of the internal operationalisation and refer only to the external specification of the competence. In the case of the domain model, such an encapsulation is not possible because task and problem-solving methods need access to meta-knowledge and domain knowledge. In the case of the task, such an encapsulation is not necessary because it does not have an internal implementation. Its implementation is described by the problem-solving method.

Formal development of knowledge-based systems requires *tool* support for modularisation of specifications and programs and for constructing, analysing and reusing proofs. We use the KIV system (Karlsruhe Interactive Verifier) (see Reif, 1995) for this purpose. Fensel and Schönegge (1997, 1998) and Fensel et al. (1998c) describe the verification of our architectural specifications using KIV. It is an advanced tool for the construction of provably correct software. KIV was originally developed for the verification of procedural programs, but it serves well for verifying knowledge-based systems. Its specification language is based on abstract data types for the functional specification of components and dynamic logic for the algorithmic specification. It provides an interactive theorem prover integrated into a sophisticated tool environment supporting aspects like the automatic generation of proof obligations, generation of counter examples, proof management, proof reuse, etc. Such support is essential for making the verification of complex specifications feasible. Currently, one has to manually translate parts of our architecture into the more general module concept of KIV. Therefore, we are working on integrating our conceptual models directly into KIV, and on proof tactics that make use of this conceptual model.

Algebraic specifications (first-order theories to be precise) were used in the project FAN (described in Renardel de Lavalette et al. (1997)) to specify the domain knowledge of a KBS for medical diagnosis in the area of anaesthetics. This formalisation of domain knowledge was done by an expert on medical information technology (he functioned as a knowledge engineer) based on the interviews with anaesthetists. A subset of this model is presented in section 4.3. Most of the anaesthetists are not able to read the formal specification directly, but they did understand and provided feedback on choices made for the formal framework. The presented architecture assists the knowledge engineer in establishing a proper framework for this type of formalisation.

The architecture we propose has been used in Fensel (1997a) and Fensel and Motta (1998) to specify libraries of PSM. The adapter concept in particular turned out to be very powerful. Usually, a PSM has many different variants. Implementing a component for each variation of a search method or each task- and domain-specific refinement is intractable. A tractable and structured approach for reusing (usable) components can only be achieved by performing refinements via adapters and implementing different aspects or degrees of refinement by different adapters. Thus, a refined method is achieved by connected to it a pile of adapters (cf. Fensel (1997a) and Fensel and Benjamins (1998a)). The architecture has been used in the course of the IBROW project (Benjamins et al., 1998) to develop the Unified Problem-solving Method development Language (UPML) (Fensel et al., 1999b) which is in the meantime being used by a number of groups for describing and interchanging problem-solving methods. An extensive case study for parametric design problem-solving is described by Motta et al. (1999). When implementing this library starting from its UPML specifications it turned out that the implementation can be done straightforwardly when using an object-oriented framework like Java. Both architectural styles match closely, which implies that most of the programming effort can be done automatically.

Acknowledgements

We thank Richard Benjamins, Stefan Decker, Gerard Renardel de Lavalette, Arno Schönegge, Remco Straatman, Rudi Studer, Annette ten Teije, Frank van Harmelen, Maarten van Someren, Bob Wielinga, Mark Willems and the anonymous reviewers for helpful comments on drafts of the paper and Jeff Butler and Nils Ramsauer for proof reading the manuscript.

References

Akkermans, JM, Wielinga, B and Schreiber, ATh, 1993. "Steps in constructing problem-solving methods" in N Aussenac et al. (eds) *Knowledge-Acquisition for Knowledge-Based Systems: Lecture Notes in AI 723* Springer-Verlag.

Angele, J, Fensel, D and Studer, R, 1998. "Developing knowledge-based systems with MIKE" *J Automated Software Engineering* **5**(4) 389–418.

Bylander, T, Allemang, D, Tanner, MC and Josephson, JR, 1991. "The computational complexity of abduction" *Artificial Intelligence* **49** 25–60.

Benjamins, R, 1995. "Problem solving methods for diagnosis and their role in knowledge acquisition" *International J Expert Systems: Research and Application* **8**(2) 93–120.

Benjamins, R, Fensel, D and Straatman, R, 1996. "Assumptions of problem-solving methods and their role in knowledge engineering" *Proc 12th European Conference on Artificial Intelligence (ECAI-96)* Budapest, Hungary, pp. 408–412.

Benjamins, VR, Plaza, E, Motta, E, Fensel, D, Studer, R, Wielinga, B, Schreiber, G and Zdrahal, Z, 1998. "An intelligent brokering service for knowledge-component reuse on the world-wide-web" *Proc 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW '98)* Banff, Canada.

Bidoit, M, Kreowski, H-J, Lescane, P, Orejas, F and Sannella, D (eds), 1991. *Algebraic System Specification and Development: Lecture Notes in Computer Science (LNCS) 501* Springer-Verlag.

Breuker, J and Van de Velde, W (eds), 1994. *The CommonKADS Library for Expertise Modelling* IOS Press.

Chandrasekaran, B, 1986. "Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design" *IEEE Expert* **1**(3) 23–30.

Chandrasekaran, B, Johnson, TR and Smith, JW, 1992. "Task structure analysis for knowledge modeling" *Communications of the ACM* **35**(9) 124–137.

Davis, R, 1984. "Diagnostic reasoning based on structure and behaviour" *Artificial Intelligence* **24** 347–410.

van Eck, P, Engelfriet, J, Fensel, D, van Harmelen, F, Venema, Y and Willems, M, 1998. "Specification of dynamics for knowledge-based systems" in B Freitag et al. (eds), *Transactions and Change in Logic Databases: Lecture Notes in Computer Science (LNCS), 1472* Springer-Verlag, pp. 37–68.

Eriksson, H, Shahar, Y, Tu, SW, Puerta, AR and Musen, MA, 1995. "Task modeling with reusable problem-solving methods" *Artificial Intelligence* **79**(2) 293–326.

Fensel, D, 1995a. "Assumptions and limitations of a problem-solving method: a case study" *Proc 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW '95)* Banff, Canada.

Fensel, D, 1995b. *The Knowledge Acquisition and Representation Language KARL* Kluwer Academic.

Fensel, D, 1995c. "Formal specification languages in knowledge and software engineering" *The Knowledge Engineering Review* **10**(4) 361–404.

Fensel, D, 1997a. "The tower-of-adapter method for developing and reusing problem-solving methods" in E Plaza et al. (eds) *Knowledge Acquisition, Modeling and Management: Lecture Notes in Artificial Intelligence (LNAI), 1319* Springer-Verlag.

Fensel, D, 1997b. "An ontology-based broker: making problem-solving method reuse work" *Proc Workshop on Problem-Solving Methods for Knowledge-based Systems at the 15th International Joint Conference on AI (IJCAI-97)* Nagoya, Japan.

Fensel, D, Angele, J and Studer, R, 1998a. "The knowledge acquisition and representation language KARL" *IEEE Trans Knowledge and Data Engineering* **10**(4) 527–550.

Fensel, D and Benjamins, VR, 1998a. "Key issues for automated problem-solving methods reuse" *Proc 13th European Conference on Artificial Intelligence (ECAI-98)* Brighton, UK.

Fensel, D and Benjamins, VR, 1998b. "The role of assumptions in knowledge engineering" *International J Intelligent Systems* **13**(8) 715–748.

Fensel, D, Benjamins, VR, Decker, S, Gaspari, M, Groenboom, R, Grosso, W, Musen, M, Motta, E, Plaza, E, Schreiber, G, Studer, R and Wielinga, B, 1999a. "The Component Model of UPML in a nutshell" *WWW Proc 1st Working IFIP Conference on Software Architectures (WICSA1)* San Antonio, TX.

Fensel, D, Eriksson, H, Musen, MA and Studer, R, 1996. "Conceptual and formal specifications of problem-solving methods" *International J Expert Systems* **9**(4) 507–532.

Fensel, D and Groenboom, R, 1996. "MLPM: Defining a semantics and axiomatization for specifying the reasoning process of knowledge-based systems" *Proc 12th European Conference on Artificial Intelligence (ECAI-96)* Budapest, Hungary.

Fensel, D, Groenboom, R and Renardel de Lavalette, GR, 1998b. "MCL: Specifying the reasoning of knowledge-based systems" *Data and Knowledge Engineering (DKE)* **26**(3) 243–269.

Fensel, D and van Harmelen, F, 1994. "A comparison of languages which operationalize and formalize KADS models of expertise" *The Knowledge Engineering Review* **9**(2) 105–146.

Fensel, D, van Harmelen, F, Reif, W and ten Teije, A, 1998c. "Formal support for development of knowledge-based systems" *Information Technology Management: An International Journal* **2**(4).

Fensel, D and Motta, E, 1998. "Structured development of problem-solving methods" *Proc 11th Workshop on Knowledge Acquisition, Modellng and Management (KAW '98)* Banff, Canada.

Fensel, D, Motta, E, Benjamins, VR, Decker, S, Gaspari, M, Groenboom, R, Grosso, W, van Harmelen, F, Musen, M, Plaza, E, Schreiber, G, Studer, R, ten Teije, A and Wielinga, B, 1999b. "The Unified Problem-solving Method development Language UPML". ESPRIT project number 27169, IBROW3, Deliverable 1.1, Chapter 1.

Fensel, D, Motta, E, Decker, S and Zdrahal, Z, 1997. "Using ontologies for defining tasks, problem-solving methods and their mappings" in E Plaza et al. (eds), *Knowledge Acquisition, Modeling and Management: Lecture Notes in Artificial Intelligence (LNAI), 1319* Springer-Verlag, pp. 113–128.

Fensel, D and Schönegge, A, 1997. "Using KIV to specify and verify architectures of knowledge-based systems" *Proc 12th IEEE International Conference on Automated Software Engineering (ASEC-97)* Incline Village, NV.

Fensel, D and Schönegge, A, 1998. "Inverse verification of problem-solving methods" *International J Human–Computer Studies* **49**(4) 339–362.

Fensel, D, Schönegge, A, Groenboom, R and Wielinga, B, 1996. "Specification and verification of knowledge-based systems" *Proc 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW '96)* Banff, Canada.

Fensel, D and Straatman, R, 1996. "The essence of problem-solving methods: Making assumptions for efficiency reasons" in N Shadbolt et al. (eds), *Advances in Knowledge Acquisition: Lecture Note in Artificial Intelligence, LNAI 1076* Springer-Verlag, pp. 17–32.

Fensel, D and Straatman, R, 1998. "The essence of problem-solving methods: Making assumptions to gain efficiency" *International J Human–Computer Studies* **48**(2) 181–215.

Gamma, E, Helm, R, Johnson, R and Vlissides, J, 1995. *Design Patterns* Addison-Wesley.

Garlan, D and Perry, D (eds), 1995. Special Issue on "Software Architecture" *IEEE Trans Software Engineering* **21**(4).

de Geus, F and Rotterdam, E, 1992. "Decision Support in Anaesthesia" PhD thesis, University of Groningen, The Netherlands.

Groenboom, R, 1997. "Formalizing Knowledge Domains – Static and Dynamic Aspects" PhD thesis, University of Groningen.

Groenboom, R and Renardel de Lavalette, GR, 1993. "Reasoning about dynamic features in specification languages" in DJ Andrews et al. (eds) *Proc Workshop in Semantics of Specification Languages* Utrecht. Springer-Verlag.

Gruber, TR, 1993. "A translation approach to portable ontology specifications" *Knowledge Acquisition* **5** 199–220.

Gurevich, Y, 1993. "Evolving Algebras 1993: Lipari Guide" in EB Börger (ed) *Specification and Valid Methods* Oxford University Press.

Harel, D, 1984. "Dynamic logic" in D Gabby et al. (eds) *Handbook of Philosophical Logic: vol. II, Extensions of Classical Logic* Kluwer.

van Harmelen, F and Aben, M, 1996. "Structure-preserving specification languages for knowledge-based systems" *J Human Computer Studies* **44** 187–212.

van Harmelen, F and Balder, J, 1995. "$(ML)^2$, a formal language for KADS conceptual models" *Knowledge Acquisition* **4**(1).

van Harmelen, F and Fensel, D, 1995. "Formal methods in knowledge engineering" *The Knowledge Engineering Review* **10**(4).

van Harmelen, F and ten Teije, A, 1998. "Characterising approximate problem-solving by partial pre- and postcondition" *Proc 13th European Conference on Artificial Intelligence (ECAI-98)* Brighton, UK.

van Heijst, G, Schreiber, AT and Wielinga, BJ, 1997. "Using explicit ontologies in knowledge-based systems development" *International J HumanComputer Interaction* **46**(6).

Kripke, SA, 1959. "A completeness theorem in modal logic" *J Symbolic Logic* **24** 1–14.

Marcus, S (ed), 1988. *Automating Knowledge Acquisition for Experts Systems* Kluwer Academic.

Menzies, T, 1999. "Knowledge maintenance: The state of the art" *The Knowledge Engineering Review* **14**(1) 1–46.

Motta, E, Gaspari, M and Fensel, D, 1999. "UPML Specification of a Parametric Design Library" Deliverable D4.1., Esprit Project 27169, IBROW3, 1999.

Motta, E and Zdrahal, Z, 1996. "Parametric design problem solving" *Proc 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW '96)* Banff, Canada.

Newell, A, 1982. "The knowledge level" *Artificial Intelligence* **18** 87–127.

Penix, J and Alexander, P, 1997. "Toward automated component adaptation" *Proc 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)* Madrid, Spain.

Penix, J, Alexander, P and Havelund, K, 1997. "Declarative specifications of software architectures" *Proc 12th IEEE International Conference on Automated Software Engineering (ASEC-97)* Incline Village, NV.

Pierret-Golbreich, C and Talon, X, 1996. "An algebraic specification of the dynamic behaviour of knowledge-based systems" *The Knowledge Engineering Review* **11**(2).

Poeck, K, Fensel, D, Landes, D and Angele, J, 1996. "Combining KARL and CRLM for designing vertical transportation systems" *International J Human–Computer Studies* **44**(3–4) 435–467.

Puppe, F, *Systematic Introduction to Exper Systems: Knowledge Representation and Problem-Solving Methods* Springer-Verlag.

Reif, W, 1992. "Correctness of generic modules" in Nerode and Taitslin (eds) *Symposium on Logical Foundations of Computer Science: LNCS 620* Springer-Verlag.

Reif, W, 1995. "The KIV approach to software engineering" in M Broy and S Jähnichen (eds) *Methods, Languages, and Tools for the Construction of Correct Software: LNCS 1009* Springer-Verlag.

Renardel de Lavalette, GR, Groenboom, R, Rotterdam, EP, van Harmelen, F, ten Teije, A and de Geus, F, 1997. "Formalisation of anaethesiology for decision support" *Artificial Intelligence in Medicine* **11**(3) 189–214.

Schreiber, ATh, Wielinga, B, Akkermans, JM, Van De Velde, W and de Hoog, R, 1994. "CommonKADS. A comprehensive methodology for KBS development" *IEEE Expert* **9**(6) 28–37.

Shaw, A, 1989. "Reasoning about time in higher level language software" *IEEE Trans Software Engineering* **15**(7) 875–889.

Shaw, M and Garlan, D, *Software Architecture: Perspectives on an Emerging Discipline* Prentice Hall.

Spee, JW and in 't Veld, L, 1994. "The semantics of $K_{BS}SF$: A language for KBS design" *Knowledge Acquisition* **6**.

Spivey, JM, 1992. *The Z Notation. A Reference Manual, 2nd ed* Prentice Hall.

Steels, L, 1990. "Components of expertise" *AI Magazine* **11**(2).

Studer, R, Benjamins, VR and Fensel, D, 1998. "Knowledge engineering: methods and principles" *Data and Knowledge Engineering* **25**(1–2).

ten Teije, A, 1997. "Automated configuration of problem solving methods in diagnosis" PhD Thesis, University of Amsterdam, The Netherlands.

Terpstra, P, van Heijst, G, Wielinga, B and Shadbolt, N, 1993. "Knowledge acquisition support through generalised directive models" in M David et al (eds) *Second Generation Expert Systems* Springer-Verlag.

Top, J and Akkermans, H, 1994. "Tasks and ontologies in engineering modeling" *International J HumanComputer Studies* **41** 585–617.

Van de Velde, W, 1988. "Inference structure as a basis for problem solving" *Proc 8th European Conference on Artificial Intelligence (ECAI-88)* Munich, Germany.

Wirsing, M, 1990. "Algebraic specification" in J van Leeuwen (ed) *Handbook of Theoretical Computer Science* Elsevier.

Yellin, DM and Strom, RE, 1997. "Protocol specifications and component adapters" *ACM Trans Programming Languages and Systems* **19**(2) 292–333.